

# Discussion of Twenty Questions Problem

Barco You

Department of Electronics and Information Engineering

Huazhong University of Science and Technology

Wuhan, China 430074

Email: barcojie@gmail.com

**Abstract**—Discuss several tricks for solving twenty question problems which in this paper is depicted as a guessing game. Player tries to find a ball in twenty boxes by asking as few questions as possible, and these questions are answered by only “Yes” or “No”. With the discussion, demonstration of source coding methods is the main concern.

## I. INTRODUCTION

Unit computation of modern computer is still binary, while “Yes or No” question is a good illustration of such computing, asking one question is equivalent to spending one bit of computation resource. This discussion is intended to give an intuition behind symbol source coding through discussing the different ways for solving a concrete twenty question problem.

The rest of this paper is organized as follows. Section II introduces the way of one-by-one asking. Section III is about top-down division. In Section IV we discuss the way of down-top merging. The work is concluded in Section V.

## II. ONE-BY-ONE ASKING

We depict the TQP(Twenty Question Problem) with 20 boxes in which only one box contains a ball, shown as figure 1. With method one, we choose arbitrarily one box and say it contain the ball, if opening the box and find there is none, equivalently answered by “No”, we get information content  $\log \frac{20}{19}$ . Continuously we draw another box but miss the ball again, we get information content  $\log \frac{19}{18}$ . Step forward repeatedly, and assume the ball is found at step  $N(1 \leq N \leq 20)$ , up to now the total information content we got is  $(\log \frac{20}{19} + \log \frac{19}{18} + \dots + \log \frac{20-N+2}{20-N+1} + \log \frac{20-N+1}{1} = \log \frac{20}{1} = 4.3219\text{bits})$ .



Fig. 1. Only one of twenty boxes includes a ball

Without loss of generality, the guessing process is illustrated as choosing the boxes in order from left to right, shown as figure 2. For every guessing, we have “Yes” or “No” results, Imagine that 1 bit is spent for every guessing. Then the expected bits need solving the TQP with the One-by-One method equals to  $(1 + \frac{19}{20} + \frac{18}{20} + \dots + \frac{2}{20} = \frac{209}{20} = 10.45\text{bits})$ .

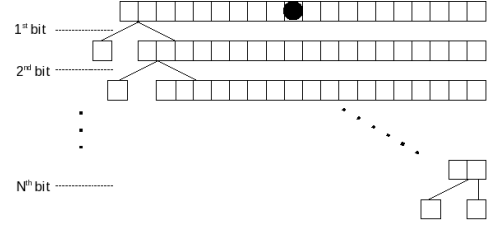


Fig. 2. Illustration of One-by-One Asking

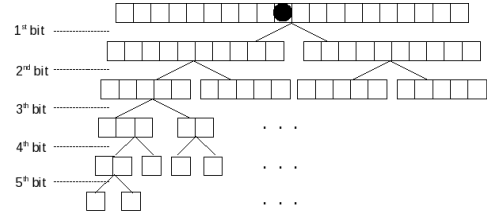


Fig. 3. Illustration of Top-Down Division

## III. TOP-DOWN DIVISION

Before every asking we divide equally the boxes into two groups, then ask if the ball is in one of the two groups. According to the answer continue this strategy repeatedly until the ball is found. This division process is shown as figure 3. In this way the expected bits to spend is  $(1 + 1 + 1 + 1 + 1 \times \frac{8}{20} = 4.4\text{bits})$ .

The information content gotten from this way is  $(1 + (\frac{10}{20} + \frac{10}{20}) + \frac{5}{20} \times (\frac{3}{5} \log \frac{5}{3} + \frac{2}{5} \log \frac{5}{2}) \times 4 + \frac{2}{20} \times 4 + \frac{3}{20} \times (\frac{2}{3} \log \frac{3}{2} + \frac{1}{3} \log 3) \times 4 + \frac{2}{20} \times 4 = \log 20 = 4.3219\text{bits})$ .

## IV. DOWN-TOP MERGING

The smartest way presented here is to merge the options in Down-Top direction, which follows Huffman Coding method [1]. Every box has the same probability  $\frac{1}{20}$  to contain the ball, combine two of the boxes and imagine they become a bigger one, then the probability of the ball in this bigger box is  $\frac{2}{20}$ . For every merging we make sure that the two boxes (real or imagined box) have the smallest probability of including the ball. For example, after first merging we have one bigger box which has probability  $\frac{2}{20}$  and there are 18 boxes with probability  $\frac{1}{20}$ , so 9 bigger boxes should be formed from the 18 boxes respectively. Repeat merging bigger boxes until we have a box which include the ball with probability 1. This merging process is shown as figure 4. From this process we have the spent bits is  $(1 + \frac{12}{20} + (\frac{8}{20} \times 2) + (\frac{4}{20} \times 5) + (\frac{2}{20} \times 10) = 4.4\text{bits})$ .

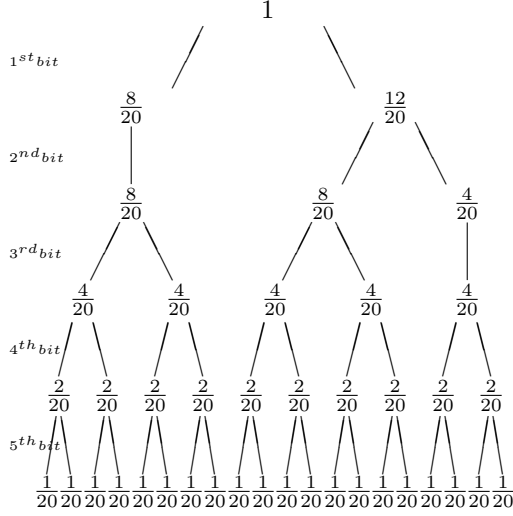


Fig. 4. Illustration of Down-Top Merging

The information content gotten in this way is  $(1 \times (\frac{8}{20} \log \frac{20}{8} + \frac{12}{20} \log \frac{20}{12}) + \frac{12}{20} \times (\frac{8}{12} \log \frac{12}{8} + \frac{4}{12} \log \frac{12}{4}) + \frac{8}{20} \times 1 \times 2 + \frac{4}{20} \times 1 \times 5 + \frac{2}{20} \times 1 \times 10 = 4.3219 \text{bits})$ .

## V. CONCLUSION

From above discussion, we can definitely conclude that to find the ball the three tricks get the same information content, but the first method consume in average much more extra effort than the later two methods. For TQP, the Top-Down Division method and Down-Top Merging method consume the same expected bits for achieving the goal. But they are not of the same efficiency. Actually the Down-Top Merging is optimal while Top-Down Division is sub-optimal, just like nuclear fusion has much more energy than nuclear fission.

**Theorem 1.** For symbol coding, Huffman code is the optimal.

*Proof:* Let symbol set  $\mathcal{A}_X = \{x_1, \dots, x_N\}$  have  $\mathcal{P}_X = \{p_1, \dots, p_N\}$ . Use division or merging method to construct codes for symbols, with once division or merging we have a new level. At any level  $I$  there are intermediate symbols  $\mathcal{A}_I = \{\alpha_1, \dots, \alpha_{n_I}\} (2 \leq n_I \leq N)$ , and  $\mathcal{P}_I = \{p_1, \dots, p_{n_I}\} (\sum_{k=1}^{n_I} p_k = 1)$ . With Huffman coding method, at level  $I$  we merge two symbols  $\alpha_i$  and  $\alpha_j$ ,  $\forall k \in \{1, \dots, n_i\}$  and  $k \neq i, k \neq j$ :  $p_k \geq p_i, p_j$ . Then the bits consumed by this merge is  $1 \times (p_i + p_j)$ . With other code, at any level  $I$  if two symbols  $\alpha_{k_1}$  and  $\alpha_{k_2}$  merge into or are divided from  $(I - 1)$  level. The consumed bits  $1 \times (p_{k_1} + p_{k_2}) \geq 1 \times (p_i + p_j)$ , if  $k_1, k_2 \neq i, j$ . Sum all the bits consumed at all levels, we can get the Huffman code is the shortest. ■

Take an example as figure 5. A symbol set with  $\mathcal{P}_X = \{\frac{2}{5}, \frac{1}{3}, \frac{1}{5}, \frac{1}{15}\}$ , with Huffman merging we get expected code length  $(1 + \frac{9}{15} + \frac{4}{15} = 1.87 \text{bits})$ , while greedy division has expected code length  $(1 + 1 = 2 \text{bits})$ .

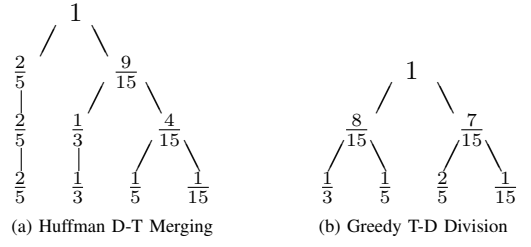


Fig. 5. Comparison between Huffman method and Greedy division

## REFERENCES

- [1] D.A Huffman, *A Method for the Construction of Minimum-Redundancy Codes*, pp 1098-1102. Proceedings of I.R.E, September, 1952.